**CMPEN 454**

**Project 1**

**Fall 2016**

Created by: Andrew Lee

Nathan Scribano

Eric Traister

1. Summary

The convolutional neural net (CNN) project was a first step into understanding fundamentals of modern computer image processing design. Implementing a forward pass CNN was a deep learning experience that stretched our understanding of object recognition from an algorithmic standpoint. The project as a whole was split into several stages of development from structural conception to layer by layer development and finally merge and design verification.

With the assistance of a debugging interface, development was easily verified in each layer of design allowing us insight into expected results. Skills attributed to and learned from in the CNN project design included many Matlab programming techniques and algorithms. Most of our team had not extensively used Matlab in the past, and as a result, we gained great experience working with the powerful language in extensive use of general multi-dimensional arrays and specific image processing and filtering techniques.

The quantitative performance evaluation gave us insight into validating and verifying the functionality of our simple CNN. We did not have any initial expectations of the neural net’s performance so the results obtained were discussed as plausible given the sophistication and depth of the CNN.

1. Outline – procedural approach, flowchart of structure

Initial development of the CNN was focused towards creation of a working CNN from the image set extracted from the ‘cifar10testdata’ data set. The procedural approach started with a function that loaded the test data, neural net filters and parameters, and the provided debugging test data for design verification.

We created separate demo functionality in the CNN to further verify the efficiency and performance of the neural net. Input images are read from the respective directories the demo is contained in. These images are then converted into working Matlab test data and pushed through a modified version of our CNN tweaked to evaluate each inputted image individually with intermediate results showing progress of “learned” features.

In our main neural net function, the image set extracted from the test data set is sized and parsed in a for loop for each image the set contains. After converting the current image to double, the image then forward prorogates through the neural net layer by layer. The CNN’s layers were structured accordingly:

Normalization Layer:

* Input is the current ‘image’. Output returns the image computed with the provided mathematical normalization calculation. No special design decisions required.

Convolution Layer:

* Input is the current ‘image’ and the CNN parameter’s filter banks along with the bias values. Output returns the convolved image by iterating over every value of the image’s array and convolving the image with each of the predefined linear filters and bias values.
* Design consisted of a double nested for loop that iterates over and convolves the image’s three channels into the filter bank and corresponding bias value’s ten channels.

Rectified Linear Unit:

* Input is the current ‘image’. Output returns the image computed with the provided mathematical ReLU calculation. No special design decisions required.

Maxpool Layer:

* Input is the current ‘image’. Output returns the image’s max value at each 2x2 sub-array to decrease the spatial dimensions of the image to a more manageable and smaller representation of the input image.
* Maxpool has to operate over each channel individually and does so by iterating through each channel with a for loop and chopping the image into four arrays with 2x2 sub arrays that are then compared to find top/bottom and left/right max values. These are then converged to find the overall max value of the top and bottom halves.

Fully Connected Layer:

* Input is the current ‘image’. Output returns an array similar to the convolution layer but limited to a 1x1xD array that contains only the image’s channel convolution properties.
* A quadruple nested for loop was used to iterate across each NxM pixel and each D dimension in the image along with each filter bank level that must be used for convolution. This step is the most time consuming layer as it must convolve each image with an equivalent size filter for every filter in the bank.

Softmax Layer:

* Input is the current ‘image’. Output returns a set of probabilities from 0 to 1 that resulted in the neural net’s evaluation of the input image.
* The design iterates across each dimension using a for loop and computes the output as the result given to us with the provided mathematical exponentiation equation to find the probabilities.

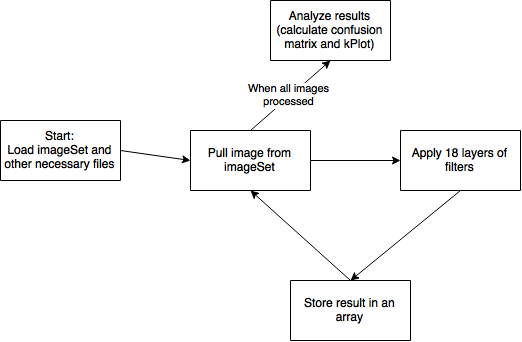
Result Analysis:

Confusion Matrix Calculation:

* + The confusion matrix is a 10x10 array, where the columns represent the correct image class and the rows represent the CNN’s guessed image class for each image in the set. The input is the results and the length of the results.
  + The function iterates across each image of the results, and adds 1 to the location of the confusion matrix where the correct class column and guessed class row intersect.
  + The output is the 10x10 array after each image’s guess is calculated.

k-Plot Calculation (ANDY):

Flowchart of Execution:



1. Experimental Observations

For the most part, the program runs as we expect it. Because many operations were vectorized, the program runs efficiently for most steps, except for fullconnect(). This is because the only way we knew to accurately implement fullconnect() was to nest a number of for loops together. As for how accurately the CNN performs, we initially thought its top guess accuracy of 43.7% seemed a little low, but after thinking about how more accurate CNNs are often composed of many more than 18 layers, we figured the accuracy isn’t too bad. When looking at the intermediate images during the process, some of the filters appear like we would expect – strongly reacting to the class’s filter – while others did not react as much as we anticipated. Intermediate images from demo1() are shown below.

Input image (32x32x3):

demo/airplane.jpg

After layer 3 (32x32x10):

demo/layer3-1.jpg demo/layer3-2.jpg demo/layer3-3.jpg demo/layer3-4.jpg demo/layer3-5.jpg demo/layer3-6.jpg demo/layer3-7.jpg demo/layer3-8.jpg demo/layer3-9.jpg demo/layer3-10.jpg

After layer 8 (16x16x10):

demo/layer8-1.jpg demo/layer8-2.jpg demo/layer8-3.jpg demo/layer8-4.jpg demo/layer8-5.jpg demo/layer8-6.jpg demo/layer8-7.jpg demo/layer8-8.jpg demo/layer8-9.jpg demo/layer8-10.jpg

After layer 13 (8x8x10):

demo/layer13-1.jpg demo/layer13-2.jpg demo/layer13-3.jpg demo/layer13-4.jpg demo/layer13-5.jpg demo/layer13-6.jpg demo/layer13-7.jpg demo/layer13-8.jpg demo/layer13-9.jpg demo/layer13-10.jpg

The output results from the softmax layer:

airplane 42.228%

automobile 1.662%

bird 0.863%

cat 0.160%

deer 0.506%

dog 0.029%

frog 0.055%

horse 0.069%

ship 53.749%

truck 0.679%

The CNN’s output percentages for the input airplane image for the most part fall in line with our expectations. The CNN seems to mistake an airplane for a ship, but that’s understandable, because they are relatively similar looking objects. It’s difficult to see how the intermediate images correlate to the final results, because the airplane and ship intermediate images don’t stick out significantly from the other intermediate class images.

1. Performance Evaluation

Collective Development:

* CNN framework
* Normalization layer
* ReLU layer
* Quantitative Evaluation

Lee’s Development:

* Softmax layer
* K-Plot

Traister’s Development:

* Fully Connected layer
* Written report

Scribano’s Development:

* Convolution layer
* Maxpool layer
* Debug function
* Confusion Matrix

1. Explorative Step
2. Contributions